ALGORITHMICS, CRYPTOGRAPHY & COMPUTER SECURITY, graduate programs 2023
CRYPTOGRAPHY, 2022, lab assignments list # 2, 27.04.2023 **updated** 05.06.2023

In this list you will work in groups of 3-4 people to implement a small, simple blockchain system. The list is incremental. You can get up to 20 points total, but there should be at least one person from each group present at each lab to report (and present) progress.

Most of the descriptions in task 1 serve as guidelines and you don't have to be too strict on following them (as long as the core philosophy is maintained). In most cases, if you find a better solution, you are welcome to implement (and defend) it.

1. (9pts total) (weeks 1-2) Design a simple blockchain system with a redundant side links.

   A blockchain is a peer-to-peer system and data structure that holds arbitrary data (records) in a sequential block structure. For the purpose of this task you may assume that the structure is static, i.e. all nodes within the structure know each other and replicate the chain fully.[1]

   Blockchain is a structure that is usually replicated on all nodes within the system and consists of blocks containing data, timestamp, and a cryptographic hash of the previous block. Blockchain with side links contains more than a single hash, instead it also features n (a parameter) hashes from other blocks, selected pseudorandomly. The additional hashes provide security against forks. By linking to pseudorandomly selected blocks in the main chain, they ensure that even if a block becomes compromised (e.g. by finding a collision), another path of trust can be established, tracing back to the original block.

   Proof of Work (PoW) is a consensus mechanism used by some blockchains (e.g. Monero, Bitcoin, until recently Ethereum) to verify and validate the state of the chain data structure. The basic idea behind proof of work is to require a certain amount of computational effort, or "work", to be performed by nodes on the network in order to add a new block of transactions. The difficulty of the puzzle is adjusted dynamically based on the total computing power of the network, so that new blocks are added to the blockchain at a consistent rate. For the sake of the task you may assume that the computing power of the network is constant (as the network is static), but the difficulty needs to be adjusted based on the actual computing power, according to a setup parameter.

   For the sake of this task, PoW will be a simple hash function. Select any secure hash function (or at least considered secure as of 2023), e.g. SHA256, Blake2b, Keccak/SHA3. We denote this hash function as H. A PoW$_d(data) = b$ with difficulty d over $data$ is a bit string b s.t.

$$token = H(H(data)||b)$$

$$token < \frac{2^l}{d}$$

(which is equivalent to:)

$$\frac{token}{2^l} < \frac{1}{d}$$

   Where l is the bit length of $token$, i.e. the output bit length of your hash function. Select an appropriate bit length of b (consider the probability that no string of this length produces an output satisfying the required property).

   If you wish, you may modify the procedure, as long as it's easy to verify (computing $H(H(data)||b)$ is easy), the difficulty d is parameterizable and it's difficult to find a collision on $data$.

[1] In principal most blockchains are dynamic and decentralized, but this would require some extra work.

Blockchain parameters The blockchain must be parametrized with at least the following parameters:

- n number of side links. Every block i past n must confirm $n + 1$ past blocks, with $i - 1$ being the first and the remaining n being selected pseudorandomly. The method of selection is described below. For blocks $0 < i \leq n$ the number of links is just $i - 1$, i.e. the blocks must link to all previous blocks.
- t the network difficulty. Expressed in seconds denotes the expected time between confirmation of consecutive blocks. Consequently, if the network hash rate is h (determined either online or as an optional parameter) the difficulty $d[H] = h[H/s] \cdot t[s]$ describes the expected number of hashes that need to be computed

Data structure Your group decides how the blockchain stores the data, presents and transmits it, but it must be consistent (across elements) in order for anyone to be able to independently validate e.g. a hash of the block. Some suggestions: normalized JSON, XML.

Each *block* is a data structure that contains the following:

index (integer, denoted i below) - number of block. The first "proper" block has index 1 and confirms an artificial block with index 0 and hash 0.

main hash (hash/bit string) - hash of block $i - 1$. As stated above, block 0 has hash 0 (all-0 bits).

extra hashes (list of hash/bit string) - a list of hashes of blocks selected according to link selection algorithm below.

PoW (bit string) - the result of a proof-of-work performed by the network on the **main hash** and **extra hashes**. Once this value is computed, the block is considered confirmed and a new block is created. The value is missing for unconfirmed blocks.

timestamp (UTC timestamp) - timestamp of computing PoW.

records (list of *record* structures) - a list of data records, added to the block.

Each *record* is a data structure that contains the following:

index (integer) - number of record within the block.

timestamp (UTC timestamp) - timestamp of adding the record into the block.

content (bit string, variable length) - encoded record content; may be arbitrary data.

API The blockchain needs to support the following operations from outside the network:

- **Add a data record to the blockchain**. The record should be added to the current block with a current timestamp. Any network node may accept the record and must propagate it to all other nodes. The network should return the block number that will contain the record, even if the block hasn't been confirmed yet (see below). Each record should have a unique identifier that should also be returned by the network. Block id may be part of the record id, i.e. it's enough to consider `(block-index, index-within-block)` as record id.
- **Retrieve a block**. The network must be able to return any already confirmed block, in full. This means all records and all hashes of previous blocks (i.e. for a block i this means hash of block $i - 1$ and all n side links). Retrieving the current block is optional.

- **Retrieve number of blocks**. The network must be able to return the number of already confirmed blocks. Consequently, if the network returns i confirmed blocks, then the current block (being filled and confirmed) is $i + 1$.

There should be at least a minimal client for these API calls. Something very simple is acceptable, e.g. a web interface or a CLI.

Algorithms The network needs to additionally perform the following procedures.

After receiving any new data record, each node computes the hash of the current block (all fields within the *block* structure) and (re-)starts the PoW algorithm.

The PoW algorithm randomly selects b strings until the PoW puzzle is solved. An example of a simple PoW using a hash funcion has been provided above.

Upon finding a solution, node propagates its solution to all other nodes within the network together with the timestamp of finding the solution. The timestamp, hash and PoW are then used to start a new block.

Warning: it may happen that node 1 finds a solution while node 2 receives a new record. All records not included within the block must be pushed to the next block. Note that a node should only reply to the request to add a record (with block id and record id) if all other nodes acknowledged the record.

Side links by default are selected as follows. A block i always links to block $i - 1$. For blocks $i \leq n$ each block includes all blocks $0 \leq j < i - 1$ as their side links. For blocks $i > n$ the block indices are selected by producing n hash values $x_j = H(x_0, j)$ where $x_0 = H(block_{i-1})$ (the hash of the previous block $i - 1$) and then converting them to n *unique* indices within $[0, i - 1)$.

This can be done, for instance, by treating the hash value x_1 as an integer and using $n_1 = x_1 \mod i - 1$ as a block index. For x_2 the procedure would be similar, but taking $n_2 = x_2 \mod i - 2$ and upon a collision (i.e. $n_2 = n_1$) taking an appropriate value from the "tail" above $i - 2$ (i.e. if $n_j = n_k$ then $n'_j = i - j + k - 1$ or so; check the math).

Grading remark your group may take -2pts penalty and implement the network in a single-node mode. The single node still needs to support the operations linked in the API section.

2. (4pts typical) (weeks 2-3) For this task your group can select any of the following subtasks. Each of them is worth 2 points:

   (a) (2pts) In a traditional blockchain in order to check correctness of a block it is required to go linearly through all intermediate blocks and verify the hash and PoW. By introducing side links instead of going block-by-block, some leaps can be done. Estimate the expected number of PoWs that need to be validated while checking a path across the full blockhain (from block 0 all the way to the newest) and for two randomly selected nodes. Check your results experimentally. The question is parametric in terms of length of the chain and the number of side links. If you elect to do task 2b, use that algorithm.

   NOTE: newer blocks confirm older blocks, not the other way. The graph is directed.

   (b) (2pts) With the presented way of selecting links, the older the block, the more incoming links it has, simply because early on there were not as many blocks to choose from; on the other hand newer blocks have much lower chance of getting incoming links. Design and test another way of selecting links (still using the same basing principle, but e.g. changing the probability distribution) to compensate for that, i.e. after some amount of time every block should have (approximately) the same expected number of incoming edges (except maybe the newest ones). The solution may be parameterized, e.g. to discourage linking to the past m nodes.

(c) (2pts) Tweak your consensus algorithm (PoW, PoS, etc) to use an algorithm that favors or discourages accelerators. For instance, Bitcoin mining is currently only viable on ASICs (application specific integrated circuits) due to its simplicity, while Ethereum for the longest time was tweaked to only be viable on GPUs. Chia blockchain does Proof of Storage that requires large amount of storage. Monero uses a RandomX algorithm that is specifically tweaked to be optimal for CPU.

Research your options. Present some benchmarks.

(d) (2pts) Inevitably, there are some problems with your blockchain. Perhaps a botched implementation, or design oversight. Implement a malicious node and connect it to the network. Try influencing it in an incorrect manner, e.g. trying to spread invalid blocks, or providing invalid proof-of-work.

After that, try fixing the issues. How many did you find? How many could you fix? How many (do you believe) are left?

You can even try splitting your group into *blue* and *red* teams with blue team defending against challenges thrown at it by the red team.

Be creative with your solutions.

3. (7pts total) (last week) Implement a simple cryptocurrency on the blockchain you have designed.

- Cryptocurrency wallets are simply public keys, with their associated private keys being the "wallet" keys.
- Each wallet should start with a 0 balance, i.e. no currency in the wallet.
- Currency can be transferred between wallets by adding records to the blockchain. A transaction record must include "transmitting wallet" (**tx** public key), a "receiving wallet" (**rx** public key), a "transaction amount" (integer), and a signature of the transmitter (signature using **tx** wallet private key) under the remaining fields (**tx** public, **rx** public, amount).
- The amount of currency in the wallet can be checked by adding all "amount" fields where the wallet was the receiver (**rx** = wallet) and subtracting all "amount" fields where the wallet was the sender (**tx** = wallet). Unfortunately, in this most basic case it means all blocks must be scanned and checked to find the amount in one wallet.

  Q: Can you do better? Try to improve the procedure. Remember you have side links!
- When a transaction record is added to the blockchain, it has to be verified against double spending (or spending more than the amount in the wallet in general). Invalid transaction records must be rejected.

  Q: What is an invalid transaction record?
- In order to introduce currency into the system, typically nodes are rewarded for participating in the network. Add public keys to the nodes and treat them as their wallets. When computing PoW, instead of using $H(data)$ use $Sign(sk, data)$ using secret key of the node, and when broadcasting a successful PoW attach the public key of the node that "mined" the block (and the signature). The node that completed the PoW (which can be verified by the signature) should be rewarded for its efforts, possibly by adding an extra transaction record (e.g. from **tx** "NETWORK" to **rx** "node addr"), or by adding another record class.

  Remember to include those records when computing wallet balance.

- In general, wallets do not need to be registered and any public key can be used as a wallet.

You can use any signature scheme you wish, preferably something short based on Elliptic Curves (like ECDSA, EdDSA or EC Schnorr).

Implement a simple CLI that allows clients to check balance of any wallet (knowing the public key) and allows fund transfer given private key file (of the sender) and wallet (public key) of the receiver. IF you really want to use a web interface, consider how to keep the private key file save (i.e. the private key material should NEVER leave the sender's machine; ideally it should even be possible to store it on a separate piece of hardware and just use it for creating a signature).

As of now you can get up to 24pts for this list, with typical max of 20pts.

/-/ Marcin Słowik